Engenharia de Serviços em Rede - Trabalho Prático 1

Streaming de áudio e vídeo a pedido e em tempo real

12 de Outubro, 2023

Grupo 48

Bernard Georges
Universidade do Minho
pg53698

José Pedro Fonte
Universidade do Minho
a91775

Vasco Oliveira
Universidade do Minho
pg54269

Resumo

O presente relatório aborda o trabalho prático desenvolvido no âmbito do estudo da pilha protocolar envolvida no streaming de áudio e vídeo. Assim sendo, o grupo utilizou várias ferramentas open-source e formatos de multimédia para explorar os diferentes empacotamentos e protocolos de transporte.

1 | Introdução

No contexto das redes de computadores, a comunicação é organizada em camadas para facilitar a transmissão eficiente de dados. Essas camadas, definidas tanto pelo Modelo OSI (Open Systems Interconnection) quanto pelo Modelo TCP/IP, proporcionam estruturas modulares para entender o processo de comunicação em redes. Cada camada tem funções específicas e comunica com as camadas adjacentes para garantir uma entrega precisa e confiável dos dados. Neste trabalho foi estudada principalmente a camada de Aplicação (Application Layer), esta camada interage diretamente com as aplicações do usuário. Ela fornece serviços de rede específicos para as necessidades das aplicações, como correio eletrônico, navegação na web e transferência de arquivos.

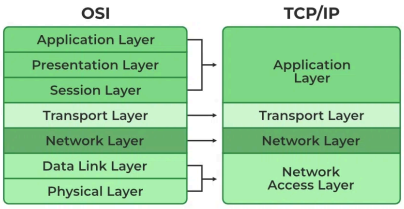


Figura 1: Modelo OSI & Modelo TCP/IP.

1.1.1 | Ferramentas Open-Source utilizadas:

- Core Network Emulator : O CORE é um emulador utilizado para construir e estudar redes virtuais.
- WireShark : O Wireshark é um programa que analisa e organiza o tráfego de rede por protocolos.
- FFmpeg : O FFmpeg é um software que grava, converte e cria stream de áudio e vídeo em diversos formatos de multimédia.
- VideoLAN VLC : O VLC media player é um software media player portátil gratuito e de código aberto, multiplataforma, e servidor de media de streaming.
- OBS Studio : O OBS é um software de gravação e transmissão ao vivo que fornece captura de fonte e dispositivo em tempo real, composição de cena, codificação, gravação e transmissão.

2 | Trabalho Desenvolvido

No arranque do trabalho prático o grupo dedicou-se à construção de uma topologia de rede, estruturando-a de acordo com os requisitos especificados no enunciado. Assim sendo, a topologia apresentada na Figura 2 serviu de base para todo o tráfego analisado.

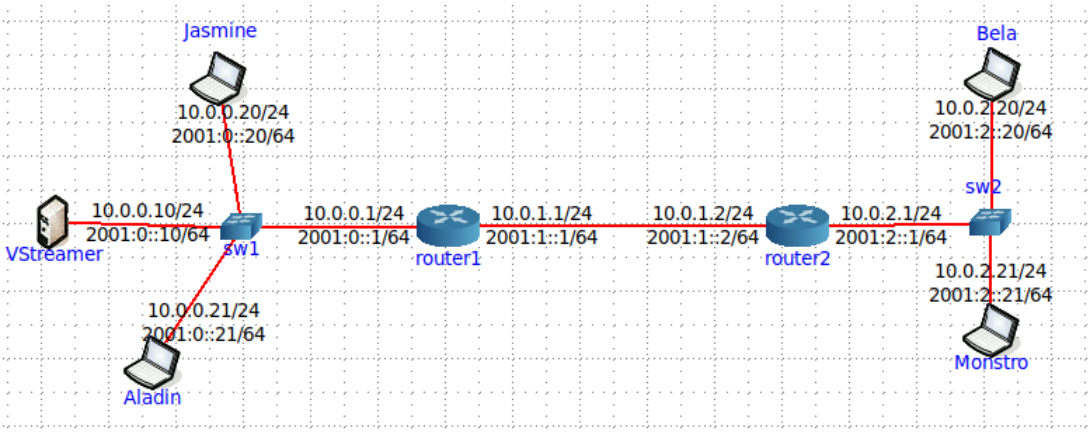


Figura 2: Topologia de Rede construída

Topologia da Rede construída e representada na Figura 2.

- **LAN 1**
  - 10.0.0.20/24 - Jasmine [PC]
  - 10.0.0.21/24 - Aladin [PC]
  - 10.0.0.10/24 - VStreamer [Servidor]
- **Routers**
  - 10.0.0.1/24 - router1 (eth0)
  - 10.0.1.1/24 - router1 (eth1)
  - 10.0.1.2/24 - router2 (eth0)
  - 10.0.2.1/24 - router2 (eth1)
- **LAN 2**
  - 10.0.2.20/24 - Bela [PC]
  - 10.0.2.21/24 - Monstro [PC]

O trabalho desenvolvido está dividido em três etapas, em conformidade com o enunciado, sendo estas:

- Etapa 1 - Streaming HTTP simples sem adaptação dinâmica de débito:

Nesta fase inicial, o foco é estabelecer um método simples de transmitir conteúdo de mídia pela internet. Não há adaptação às condições da rede, tornando a entrega de media mais estática.

- Etapa 2 - Streaming adaptativo sobre HTTP (MPEG-DASH):

Na segunda etapa, o projeto avança para o uso do MPEG-DASH, um padrão que permite que o cliente ajuste dinamicamente a qualidade do vídeo com base na largura de banda disponível.

- Etapa 3 - Streaming RTP/RTCP unicast sobre UDP e multicast com anúncios SAP:

Nesta fase mais avançada, o projeto envolve a implementação de streaming em tempo real (RTP) e seu protocolo de controle (RTCP) para comunicação direta com destinatários individuais (unicast) e também explora a entrega multicast (um-para-muitos) com a inclusão de anúncios SAP.

2.1 | Etapa 1 - Streaming HTTP simples sem adaptação dinâmica de débito

A etapa 1 teve como objetivo estudar o streaming por HTTP simples sem adaptação do débito. Nesta etapa o VLC é usado simultaneamente como servidor de streaming e como cliente. Depois são acrescentados mais dois clientes (firefox e ffplay) e estuda-se o impacto no tráfego da rede.

Questão 1 - Capture três pequenas amostras de trágefo no link de saída do servidor, respetivamente com 1 cliente (VLC), com 2 clientes (VLC e Firefox) e com 3 clientes (VLC, Firefox e ffplay).

O grupo conduziu o trabalho de maneira sequencial e cumulativa, o que implica que clientes foram adicionados gradualmente e a rede foi analisada passo a passo. De notar que todas as capturas de tela foram obtidas do servidor *VStreamer*.

O primeiro passo foi iniciar o streaming entre o VLC Server no *VStreamer* e o VLC Client na *Jasmine* , que a Figura 3 mostra.

No início, é evidente a presença de tráfego contendo pacotes *OSPF* e *ARP*. Conforme ensinado na disciplina de Redes de Computadores, o OSPF (Open Shortest Path First) desempenha o papel de um protocolo de roteamento interno, responsável por calcular as rotas mais eficientes em redes IP. Por sua vez, o ARP (Address Resolution Protocol) tem a função de mapear endereços IP nos correspondentes endereços MAC em redes locais, possibilitando a comunicação entre dispositivos dentro da mesma sub-rede.

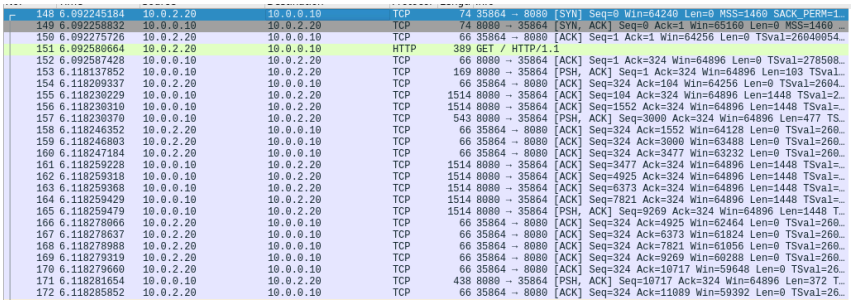
Nas linhas seguintes, podemos observar o tráfego entre os endereços IP 10.0.0.10 (*VStreamer*) e 10.0.0.20 (*Jasmine*). Esse tráfego inicia com um procedimento de *3-way handshake*, que estabelece a conexão TCP, seguido por um pedido HTTP (GET Request) para receber os pacotes de vídeo. Nas linhas subsequentes, ocorre a transferência contínua de pacotes da stream por meio do canal de comunicação criado entre o Servidor e o Cliente.

No.	Time	Source	Destination	Protocol	Length	Info
25	37.085409879	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
26	39.085768113	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
27	39.897973535	00:00:00_aa:00:01	Broadcast	ARP	42	Who has 10.0.0.10? Tell 10.0.0.20
28	39.897993759	00:00:00_aa:00:00	00:00:00_aa:00:01	ARP	42	10.0.0.10 is at 00:00:00:aa:00:00
29	39.897998239	10.0.0.20	10.0.0.10	TCP	78	35706 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1…
30	39.898008441	10.0.0.10	10.0.0.20	TCP	74	8080 → 35706 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 …
31	39.898016970	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=42030044…
32	39.898802377	10.0.0.20	10.0.0.10	HTTP	200	GET / HTTP/1.1
33	39.898808851	10.0.0.10	10.0.0.20	TCP	66	8080 → 35706 [ACK] Seq=1 Ack=135 Win=65152 Len=0 TSval=316444…
34	39.920158702	10.0.0.10	10.0.0.20	TCP	169	8080 → 35706 [PSH, ACK] Seq=1 Ack=135 Win=65152 Len=103 TSval…
35	39.920189610	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=104 Win=64256 Len=0 TSval=4203…
36	39.920241764	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=104 Ack=135 Win=65152 Len=1448 TSval=3…
37	39.920241874	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=1552 Ack=135 Win=65152 Len=1448 TSval=…
38	39.920241934	10.0.0.10	10.0.0.20	TCP	543	8080 → 35706 [PSH, ACK] Seq=3000 Ack=135 Win=65152 Len=477 TS…
39	39.920251535	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=1552 Win=64128 Len=0 TSval=420…
40	39.920251996	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=3000 Win=63488 Len=0 TSval=420…
41	39.920252347	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=3477 Win=63232 Len=0 TSval=420…
42	39.920262329	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=3477 Ack=135 Win=65152 Len=1448 TSval=…
43	39.920262419	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=4925 Ack=135 Win=65152 Len=1448 TSval=…
44	39.920262469	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=6373 Ack=135 Win=65152 Len=1448 TSval=…
45	39.920262529	10.0.0.10	10.0.0.20	TCP	401	8080 → 35706 [PSH, ACK] Seq=7821 Ack=135 Win=65152 Len=335 TS…
46	39.920268071	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=4925 Win=62464 Len=0 TSval=420…
47	39.920268482	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=6373 Win=61824 Len=0 TSval=420…
48	39.920268823	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=7821 Win=61056 Len=0 TSval=420…
49	39.920269133	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=8156 Win=60928 Len=0 TSval=420…
50	40.045229318	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=8156 Ack=135 Win=65152 Len=1448 TSval=…
51	40.045230080	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=9604 Ack=135 Win=65152 Len=1448 TSval=…
52	40.045230140	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=11052 Ack=135 Win=65152 Len=1448 TSval…
53	40.045230190	10.0.0.10	10.0.0.20	TCP	321	8080 → 35706 [PSH, ACK] Seq=12500 Ack=135 Win=65152 Len=255 T…
54	40.045256177	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=9604 Win=64128 Len=0 TSval=420…
55	40.045256939	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=11052 Win=63488 Len=0 TSval=42…
56	40.045257320	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=12500 Win=62848 Len=0 TSval=42…
57	40.045257660	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=135 Ack=12755 Win=62720 Len=0 TSval=42…
58	40.292316859	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=12755 Ack=135 Win=65152 Len=1448 TSval…
59	40.292317842	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=14203 Ack=135 Win=65152 Len=1448 TSval…

Figura 3: Captura Wireshark 1 Cliente (VStreamer -> Jasmine)

Avançando para o próximo passo, ao incorporar um novo cliente Firefox na *Bela* (IP: 10.0.2.20), torna-se relevante examinar como o servidor distribui pacotes da mesma stream para dois clientes distintos.

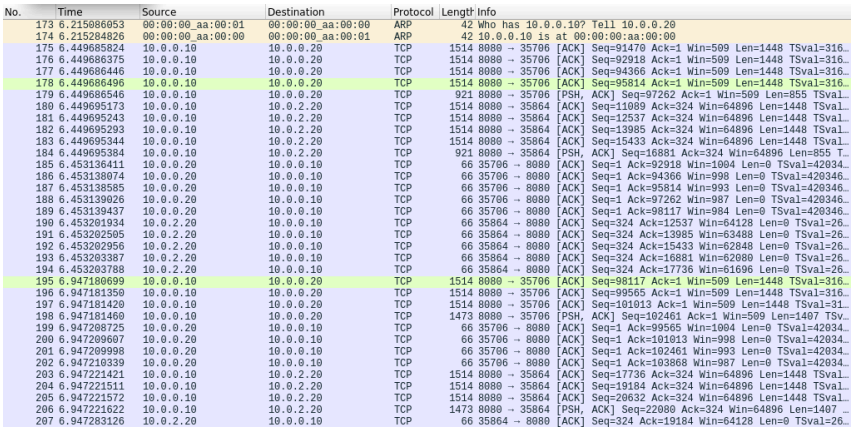
Na Figura 4 é possível observar a adição do novo cliente que, há semelhança do anterior, também estabelece um canal de comunicação através de um *3-way handshake*, seguindo-se um pedido HTTP (GET Request) para receber os pacotes de dados da stream.



148	6.092245184	10.0.2.20	10.0.0.10	TCP	74	35864 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1…
149	6.092258832	10.0.0.10	10.0.2.20	TCP	74	8080 → 35864 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 …
150	6.092275726	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=26040054…
151	6.092580664	10.0.2.20	10.0.0.10	HTTP	389	GET / HTTP/1.1
152	6.092587428	10.0.0.10	10.0.2.20	TCP	66	8080 → 35864 [ACK] Seq=1 Ack=324 Win=64896 Len=0 TSval=278508…
153	6.118137852	10.0.0.10	10.0.2.20	TCP	169	8080 → 35864 [PSH, ACK] Seq=1 Ack=324 Win=64896 Len=103 TSval…
154	6.118209337	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=104 Win=64256 Len=0 TSval=2604…
155	6.118230229	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=104 Ack=324 Win=64896 Len=1448 TSval=2…
156	6.118230310	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=1552 Ack=324 Win=64896 Len=1448 TSval=…
157	6.118230370	10.0.0.10	10.0.2.20	TCP	543	8080 → 35864 [PSH, ACK] Seq=3000 Ack=324 Win=64896 Len=477 TS…
158	6.118246352	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=1552 Win=64128 Len=0 TSval=260…
159	6.118246803	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=3000 Win=63488 Len=0 TSval=260…
160	6.118247184	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=3477 Win=63232 Len=0 TSval=260…
161	6.118259228	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=3477 Ack=324 Win=64896 Len=1448 TSval=…
162	6.118259318	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=4925 Ack=324 Win=64896 Len=1448 TSval=…
163	6.118259368	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=6373 Ack=324 Win=64896 Len=1448 TSval=…
164	6.118259429	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=7821 Ack=324 Win=64896 Len=1448 TSval=…
165	6.118259479	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [PSH, ACK] Seq=9269 Ack=324 Win=64896 Len=1448 T…
166	6.118278066	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=4925 Win=62464 Len=0 TSval=260…
167	6.118278637	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=6373 Win=61824 Len=0 TSval=260…
168	6.118278988	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=7821 Win=61056 Len=0 TSval=260…
169	6.118279319	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=9269 Win=60288 Len=0 TSval=260…
170	6.118279660	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=10717 Win=59648 Len=0 TSval=26…
171	6.118281654	10.0.0.10	10.0.2.20	TCP	438	8080 → 35864 [PSH, ACK] Seq=10717 Ack=324 Win=64896 Len=372 T…
172	6.118285852	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=11089 Win=59392 Len=0 TSval=26…

Figura 4: Captura Wireshark 2 Clientes (VStreamer -> Bela)

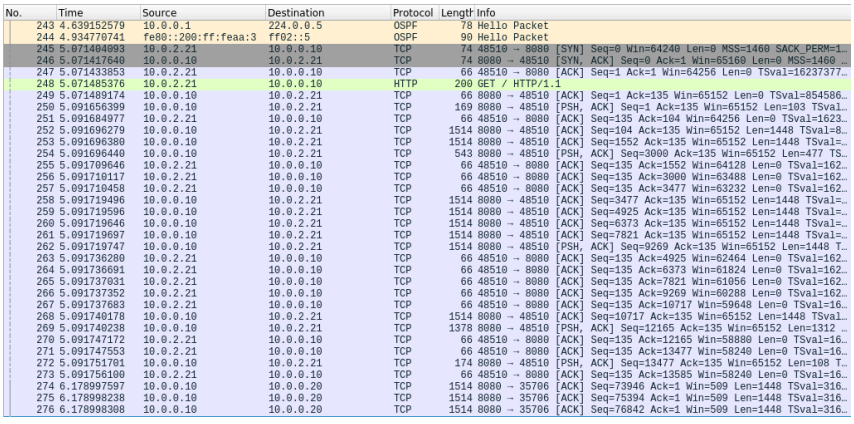
Na Figura 5, é possível notar a retransmissão de dados para o cliente *Jasmine*, que, de forma alternada com o cliente *Bela*, recebe todos os pacotes de dados da stream. Essa alternância ocorre devido ao facto do servidor direcionar todo o tráfego por apenas uma interface, o que implica a implementação de controle de fluxo para atender a todos os clientes.



No.	Time	Source	Destination	Protocol	Length	Info
173	6.215086053	00:00:00_aa:00:01	00:00:00_aa:00:00	ARP	42	Who has 10.0.0.10? Tell 10.0.0.20
174	6.215284826	00:00:00_aa:00:00	00:00:00_aa:00:01	ARP	42	10.0.0.10 is at 00:00:00:aa:00:00
175	6.449685824	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=91470 Ack=1 Win=509 Len=1448 TSval=316…
176	6.449686375	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=92918 Ack=1 Win=509 Len=1448 TSval=316…
177	6.449686446	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=94366 Ack=1 Win=509 Len=1448 TSval=316…
178	6.449686496	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=95814 Ack=1 Win=509 Len=1448 TSval=316…
179	6.449686546	10.0.0.10	10.0.0.20	TCP	921	8080 → 35706 [PSH, ACK] Seq=97262 Ack=1 Win=509 Len=855 TSval…
180	6.449695173	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=11089 Ack=324 Win=64896 Len=1448 TSval…
181	6.449695243	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=12537 Ack=324 Win=64896 Len=1448 TSval…
182	6.449695293	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=13985 Ack=324 Win=64896 Len=1448 TSval…
183	6.449695344	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=15433 Ack=324 Win=64896 Len=1448 TSval…
184	6.449695384	10.0.0.10	10.0.2.20	TCP	921	8080 → 35864 [PSH, ACK] Seq=16881 Ack=324 Win=64896 Len=855 T…
185	6.453136411	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=92918 Win=1004 Len=0 TSval=42034…
186	6.453138074	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=94366 Win=998 Len=0 TSval=420346…
187	6.453138585	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=95814 Win=993 Len=0 TSval=420346…
188	6.453139026	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=97262 Win=987 Len=0 TSval=420346…
189	6.453139437	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=98117 Win=984 Len=0 TSval=420346…
190	6.453201934	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=12537 Win=64128 Len=0 TSval=26…
191	6.453202505	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=13985 Win=63488 Len=0 TSval=26…
192	6.453202956	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=15433 Win=62848 Len=0 TSval=26…
193	6.453203387	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=16881 Win=62080 Len=0 TSval=26…
194	6.453203788	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=17736 Win=61696 Len=0 TSval=26…
195	6.947180699	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=98117 Ack=1 Win=509 Len=1448 TSval=316…
196	6.947181350	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=99565 Ack=1 Win=509 Len=1448 TSval=316…
197	6.947181420	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=101013 Ack=1 Win=509 Len=1448 TSval=31…
198	6.947181460	10.0.0.10	10.0.0.20	TCP	1473	8080 → 35706 [PSH, ACK] Seq=102461 Ack=1 Win=509 Len=1407 TSv…
199	6.947208725	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=99565 Win=1004 Len=0 TSval=42034…
200	6.947209607	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=101013 Win=998 Len=0 TSval=42034…
201	6.947209998	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=102461 Win=993 Len=0 TSval=42034…
202	6.947210339	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=103868 Win=987 Len=0 TSval=42034…
203	6.947221421	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=17736 Ack=324 Win=64896 Len=1448 TSval…
204	6.947221511	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=19184 Ack=324 Win=64896 Len=1448 TSval…
205	6.947221572	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=20632 Ack=324 Win=64896 Len=1448 TSval…
206	6.947221622	10.0.0.10	10.0.2.20	TCP	1473	8080 → 35864 [PSH, ACK] Seq=22080 Ack=324 Win=64896 Len=1407 …
207	6.947283126	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=324 Ack=19184 Win=64128 Len=0 TSval=26…

Figura 5: Captura Wireshark 2 Clientes (VStreamer -> Jasmine & Bela)

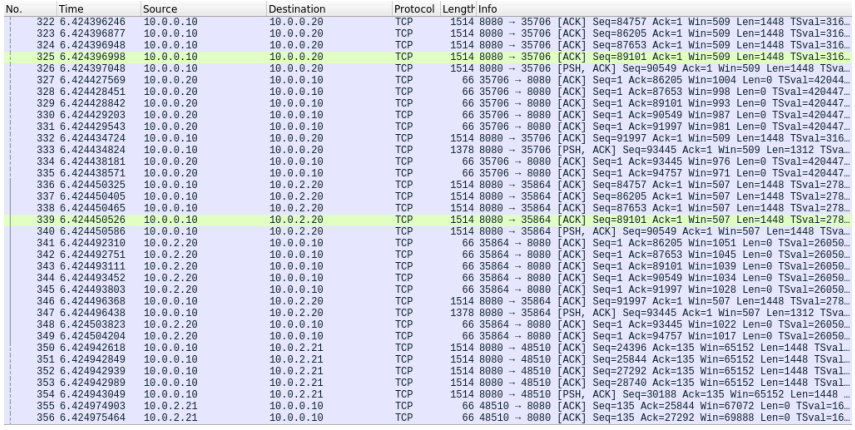
Com a adição de um novo cliente, desta vez no Monstro (IP: 10.0.2.21) e usando ffplay, as etapas anteriores se repetem. Na Figura 6, podemos notar o procedimento de *3-way handshake*, no qual se estabelece a conexão, seguido por um pedido HTTP (GET Request) e a subsequente transmissão de dados.



No.	Time	Source	Destination	Protocol	Length	Info
243	4.639152579	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
244	4.934770741	fe80::200:ff:feaa:3	ff02::5	OSPF	90	Hello Packet
245	5.071404893	10.0.2.21	10.0.0.10	TCP	74	48510 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1…
246	5.071417640	10.0.0.10	10.0.2.21	TCP	74	8080 → 48510 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 …
247	5.071433853	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=16237377…
248	5.071485376	10.0.2.21	10.0.0.10	HTTP	200	GET / HTTP/1.1
249	5.071489174	10.0.0.10	10.0.2.21	TCP	66	8080 → 48510 [ACK] Seq=1 Ack=135 Win=65152 Len=0 TSval=854586…
250	5.091656399	10.0.0.10	10.0.2.21	TCP	169	8080 → 48510 [PSH, ACK] Seq=1 Ack=135 Win=65152 Len=103 TSval…
251	5.091684977	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=104 Win=64256 Len=0 TSval=1623…
252	5.091696279	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=104 Ack=135 Win=65152 Len=1448 TSval=8…
253	5.091696380	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=1552 Ack=135 Win=65152 Len=1448 TSval=…
254	5.091696440	10.0.0.10	10.0.2.21	TCP	543	8080 → 48510 [PSH, ACK] Seq=3000 Ack=135 Win=65152 Len=477 TS…
255	5.091709646	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=1552 Win=64128 Len=0 TSval=162…
256	5.091710117	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=3000 Win=63488 Len=0 TSval=162…
257	5.091719458	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=3477 Win=63232 Len=0 TSval=162…
258	5.091719496	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=3477 Ack=135 Win=65152 Len=1448 TSval=…
259	5.091719596	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=4925 Ack=135 Win=65152 Len=1448 TSval=…
260	5.091719646	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=6373 Ack=135 Win=65152 Len=1448 TSval=…
261	5.091719697	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=7821 Ack=135 Win=65152 Len=1448 TSval=…
262	5.091719747	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [PSH, ACK] Seq=9269 Ack=135 Win=65152 Len=1448 T…
263	5.091736280	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=4925 Win=62464 Len=0 TSval=162…
264	5.091736691	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=6373 Win=61824 Len=0 TSval=162…
265	5.091737031	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=7821 Win=61056 Len=0 TSval=162…
266	5.091737352	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=9269 Win=60288 Len=0 TSval=162…
267	5.091737683	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=10717 Win=59648 Len=0 TSval=16…
268	5.091740178	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=10717 Ack=135 Win=65152 Len=1448 TSval…
269	5.091740238	10.0.0.10	10.0.2.21	TCP	1378	8080 → 48510 [PSH, ACK] Seq=12165 Ack=135 Win=65152 Len=1312 …
270	5.091747172	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=12165 Win=58880 Len=0 TSval=16…
271	5.091747553	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=13477 Win=58240 Len=0 TSval=16…
272	5.091751701	10.0.0.10	10.0.2.21	TCP	174	8080 → 48510 [PSH, ACK] Seq=13477 Ack=135 Win=65152 Len=108 T…
273	5.091756100	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=13585 Win=58240 Len=0 TSval=16…
274	6.178997597	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=73946 Ack=1 Win=509 Len=1448 TSval=316…
275	6.178998238	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=75394 Ack=1 Win=509 Len=1448 TSval=316…
276	6.178998308	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=76842 Ack=1 Win=509 Len=1448 TSval=316…

Figura 6: Captura Wireshark 3 Clientes (VStreamer -> Monstro)

Na Figura 7, verifica-se novamente a comunicação intercalada com os vários clientes, já descrita e explicada anteriormente.



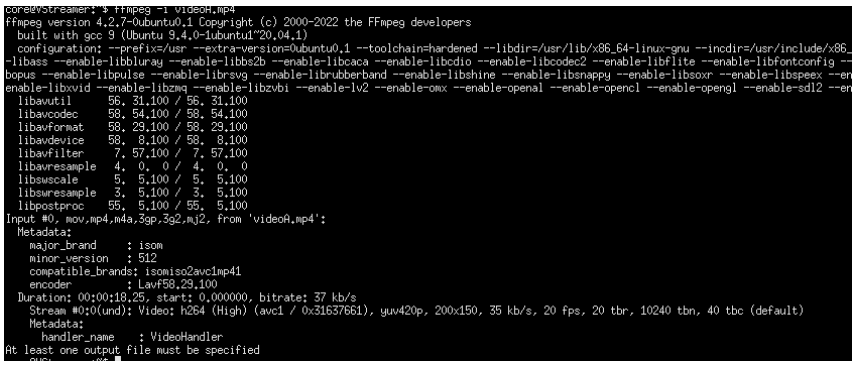
No.	Time	Source	Destination	Protocol	Length	Info
322	6.424396246	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=84757 Ack=1 Win=509 Len=1448 TSval=316…
323	6.424396877	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=86205 Ack=1 Win=509 Len=1448 TSval=316…
324	6.424396948	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=87653 Ack=1 Win=509 Len=1448 TSval=316…
325	6.424396998	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=89101 Ack=1 Win=509 Len=1448 TSval=316…
326	6.424397048	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [PSH, ACK] Seq=90549 Ack=1 Win=509 Len=1448 TSva…
327	6.424427569	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=86205 Win=1004 Len=0 TSval=42044…
328	6.424428451	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=87653 Win=998 Len=0 TSval=420447…
329	6.424428842	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=89101 Win=993 Len=0 TSval=420447…
330	6.424429203	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=90549 Win=987 Len=0 TSval=420447…
331	6.424429543	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=91997 Win=981 Len=0 TSval=420447…
332	6.424434724	10.0.0.10	10.0.0.20	TCP	1514	8080 → 35706 [ACK] Seq=91997 Ack=1 Win=509 Len=1448 TSval=316…
333	6.424434824	10.0.0.10	10.0.0.20	TCP	1378	8080 → 35706 [PSH, ACK] Seq=93445 Ack=1 Win=509 Len=1312 TSva…
334	6.424438181	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=93445 Win=976 Len=0 TSval=420447…
335	6.424438571	10.0.0.20	10.0.0.10	TCP	66	35706 → 8080 [ACK] Seq=1 Ack=94757 Win=971 Len=0 TSval=420447…
336	6.424450325	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=84757 Ack=1 Win=507 Len=1448 TSval=278…
337	6.424450405	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=86205 Ack=1 Win=507 Len=1448 TSval=278…
338	6.424450465	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=87653 Ack=1 Win=507 Len=1448 TSval=278…
339	6.424450526	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=89101 Ack=1 Win=507 Len=1448 TSval=278…
340	6.424450586	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [PSH, ACK] Seq=90549 Ack=1 Win=507 Len=1448 TSva…
341	6.424492310	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=1 Ack=86205 Win=1051 Len=0 TSval=26050…
342	6.424492751	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=1 Ack=87653 Win=1045 Len=0 TSval=26050…
343	6.424493111	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=1 Ack=89101 Win=1039 Len=0 TSval=26050…
344	6.424493452	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=1 Ack=90549 Win=1034 Len=0 TSval=26050…
345	6.424493803	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=1 Ack=91997 Win=1028 Len=0 TSval=26050…
346	6.424496368	10.0.0.10	10.0.2.20	TCP	1514	8080 → 35864 [ACK] Seq=91997 Ack=1 Win=507 Len=1448 TSval=278…
347	6.424496438	10.0.0.10	10.0.2.20	TCP	1378	8080 → 35864 [PSH, ACK] Seq=93445 Ack=1 Win=507 Len=1312 TSva…
348	6.424503823	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=1 Ack=93445 Win=1022 Len=0 TSval=26050…
349	6.424504204	10.0.2.20	10.0.0.10	TCP	66	35864 → 8080 [ACK] Seq=1 Ack=94757 Win=1017 Len=0 TSval=26050…
350	6.424942618	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=24396 Ack=135 Win=65152 Len=1448 TSval…
351	6.424942849	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=25844 Ack=135 Win=65152 Len=1448 TSval…
352	6.424942939	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=27292 Ack=135 Win=65152 Len=1448 TSval…
353	6.424942989	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [ACK] Seq=28740 Ack=135 Win=65152 Len=1448 TSval…
354	6.424943049	10.0.0.10	10.0.2.21	TCP	1514	8080 → 48510 [PSH, ACK] Seq=30188 Ack=135 Win=65152 Len=1448 …
355	6.424974903	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=25844 Win=67072 Len=0 TSval=16…
356	6.424975464	10.0.2.21	10.0.0.10	TCP	66	48510 → 8080 [ACK] Seq=135 Ack=27292 Win=69888 Len=0 TSval=16…

Figura 7: Captura Wireshark 3 Clientes (VStreamer -> Jasmine & Bela & Monstro)

Identifique a taxa em *bps* necessária (usando o ffmpeg -i videoA.mp4 e/ou o próprio wireshark), o encapsulamento usado e o número total de fluxos gerados.

Taxa de bps necessária

Através da Figura 8 recolhemos a informação que o *bps* mínimo para a transmissão do `videoA.mp4` é de 37 kb/s.

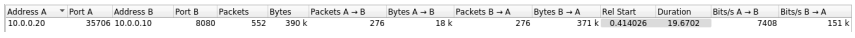


```
core@vStreamer:~$ ffmpeg -i videoA.mp4
ffmpeg version 4.2.7-0ubuntu0.1 Copyright (c) 2000-2022 the FFmpeg developers
  built with gcc 9 (Ubuntu 9.4.0-1ubuntu1~20.04.1)
  configuration: --prefix=/usr --extra-version=0ubuntu0.1 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/include/x86_
-libass --enable-libbluray --enable-libbs2b --enable-libcaca --enable-libcdio --enable-libcodec2 --enable-libflite --enable-libfontconfig --
bopus --enable-libpulse --enable-librsvg --enable-librubberband --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex --en
enable-libxvid --enable-libzmq --enable-libzvbi --enable-lv2 --enable-omx --enable-openal --enable-opencl --enable-opengl --enable-sdl2 --en
  libavutil      56. 31.100 / 56. 31.100
  libavcodec     58. 54.100 / 58. 54.100
  libavformat    58. 29.100 / 58. 29.100
  libavdevice    58.  8.100 / 58.  8.100
  libavfilter     7. 57.100 /  7. 57.100
  libavresample   4.  0.  0 /  4.  0.  0
  libswscale      5.  5.100 /  5.  5.100
  libswresample   3.  5.100 /  3.  5.100
  libpostproc    55.  5.100 / 55.  5.100
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'videoA.mp4':
  Metadata:
    major_brand     : isom
    minor_version   : 512
    compatible_brands: isomiso2avc1mp41
    encoder         : Lavf58.29.100
  Duration: 00:00:18.25, start: 0.000000, bitrate: 37 kb/s
    Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 200x150, 35 kb/s, 20 fps, 20 tbr, 10240 tbn, 40 tbc (default)
    Metadata:
      handler_name    : VideoHandler
At least one output file must be specified
```

Figura 8: Resultado do comando "`ffmpeg -i videoA.mp4`"

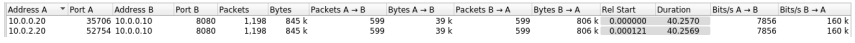
Através da Figura 9, Figura 10 e Figura 11, identificamos que a taxa em *bps* necessária é mais alta na prática, visto que os pacotes de dados incluem o *payload* e o *overhead* dos protocolos. Assim sendo, as nossas análises são:

- Com um cliente (VLC), a taxa em bps foi de 151.000 bps.
- Com dois clientes (VLC e Firefox), a taxa em bps foi de 160.000 bps para cada cliente, somando obtemos 320.000 bps (160.000 bps x 2 clientes).
- Com três clientes (VLC, Firefox e ffplay), a taxa em bps foi de 148.000 bps para cada cliente, somando obtemos 444.000 bps (148.000 bps x 3 clientes).



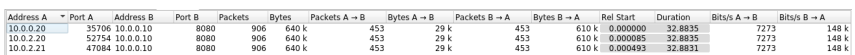
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.0.20	35706	10.0.0.10	8080	552	390 k	276	18 k	276	371 k	0.414026	19.6702	7408	151 k

Figura 9: Conexão com 1 cliente



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.0.20	35706	10.0.0.10	8080	1,198	845 k	599	39 k	599	806 k	0.000000	40.2570	7856	160 k
10.0.2.20	52754	10.0.0.10	8080	1,198	845 k	599	39 k	599	806 k	0.000121	40.2569	7856	160 k

Figura 10: Conexão com 2 clientes



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.0.20	35706	10.0.0.10	8080	906	640 k	453	29 k	453	610 k	0.000000	32.8835	7273	148 k
10.0.2.20	52754	10.0.0.10	8080	906	640 k	453	29 k	453	610 k	0.000085	32.8835	7273	148 k
10.0.2.21	47084	10.0.0.10	8080	906	640 k	453	29 k	453	610 k	0.000493	32.8831	7273	148 k

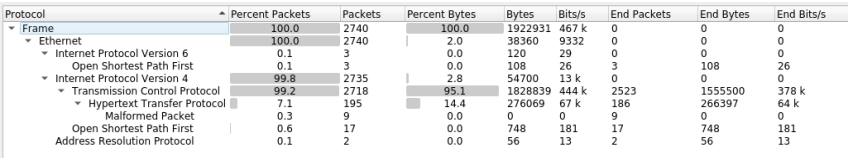
Figura 11: Conexão com 3 clientes

Esses valores representam a largura de banda total necessária para acomodar as transmissões simultâneas de todos os clientes, demonstrando uma escalabilidade linear, apesar de não totalmente perfeita, na adição de cada cliente.

Encapsulamento dos Pacotes

Quanto ao encapsulamento dos pacotes de dados, a Figura 12 evidencia os protocolos de todos os pacotes capturados. Retiramos as seguintes conclusões:

- Todos os pacotes correm sobre protocolo Ethernet, que corresponde à Camada de Ligação de Dados.
- Os pacotes ARP correm dentro da Camada de Ligação de Dados.
- Os pacotes OSPF correm em cima de IPV4 e IPV6, logo correm na Camada de Rede.
- Os pacotes de transmissão correm em TCP/IPV4 e Ethernet, logo correm na Camada de Transporte.
- Os pedidos HTTP correm na camada de aplicação, em cima de TCP,IPV4 e Ethernet.



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	2740	100.0	1922931	467 k	0	0	0
Ethernet	100.0	2740	2.0	38360	9332	0	0	0
Internet Protocol Version 6	0.1	3	0.0	120	29	0	0	0
Open Shortest Path First	0.1	3	0.0	108	26	3	108	26
Internet Protocol Version 4	99.8	2735	2.8	54700	13 k	0	0	0
Transmission Control Protocol	99.2	2718	95.1	1828839	444 k	2523	1555500	378 k
Hypertext Transfer Protocol	7.1	195	14.4	276069	67 k	186	266397	64 k
Malformed Packet	0.3	9	0.0	0	0	9	0	0
Open Shortest Path First	0.6	17	0.0	748	181	17	748	181
Address Resolution Protocol	0.1	2	0.0	56	13	2	56	13

Figura 12: Pilha Protocolar envolvida no Streaming Simples por HTTP

Fluxos Gerados

Observando a Figura 9, Figura 10 e Figura 11, obtivemos a seguinte análise em relação aos fluxos gerados:

- Com um cliente (VLC), um único fluxo de dados foi gerado entre o servidor VStreamer e o PC Jasmine.
- Com dois clientes (VLC e Firefox), dois fluxos distintos de dados foram gerados, entre o servidor VStreamer e os PCs Jasmine e Bela.
- Com três clientes (VLC, Firefox e ffplay), observamos a geração de três fluxos de dados distintos, entre o servidor VStreamer e os PCs Jasmine, Bela e Monstro.

Essa análise evidenciou que o número de fluxos gerados correspondeu diretamente ao número de clientes envolvidos em cada cenário. Esses fluxos representam as conexões de comunicação ativas entre o servidor e cada cliente.

Comente a escalabilidade da solução.

Para o estudo da escalabilidade da solução, o grupo analisou as conexões e construi a seguinte ilustração na Figura 13, que representa o que acontece na realidade.

A natureza do protocolo utilizado para a transmissão dos pacotes faz com que para cada novo cliente se crie uma nova ligação orientada à conexão, ou seja, uma nova porta onde se executa todo controlo associado ao protocolo TCP. Esta implementação implica negativamente a escalabilidade e eficiência da transmissão por vários motivos para além do mencionado, sendo os principais:

- o overhead do http torna o ínicio da conexão mais lento.
- a transmissão orientada à conexão transmite muitas vezes os mesmos dados, ao mesmo tempo.

No entanto, o que o protocolo tira em performance e escalabilidade dá em experiência de utilizador, permitindo a cada cliente navegar qualquer parte do vídeo e obter os todos pacotes de forma confiável.

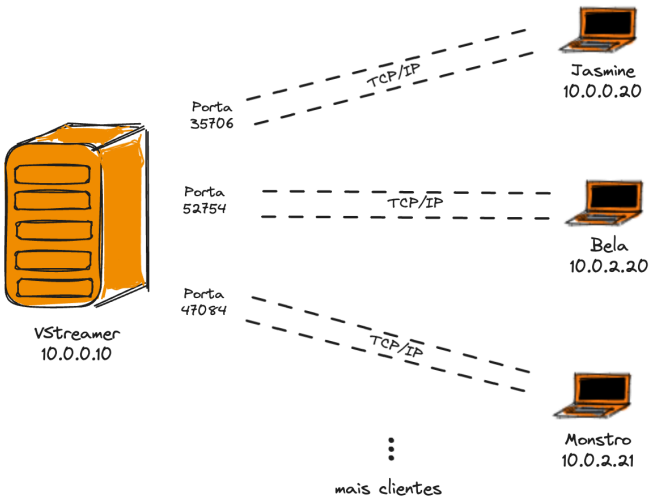


Figura 13: Fluxos Gerados entre Servidor e Clientes

2.2 | Etapa 2 - Streaming adaptativo sobre HTTP (MPEG-DASH)

O objetivo desta etapa é utilizar o *videoB.mp4* como entrada e gerar pelo menos três variantes com três resoluções diferentes, para que depois possa ser servido em streaming com débito adaptativo e estudado pelo grupo.

Questão 2 - Diga qual a largura de banda necessária, em bits por segundo, para que o cliente de streaming consiga receber o vídeo no firefox e qual a pilha protocolar usada neste cenário.

O grupo criou as três variantes do *videoB.mp4* com as suas características descritas no ficheiro `video_manifest.mpd`, detalhadas em maior detalhe na Figura 14, Figura 15 e Figura 16. Analisando o parâmetro *bandwidth*, concluímos que o mínimo de largura de banda de cada vídeo é:

- videoB200k.mp4 : 173339 *bps*
- videoB500k.mp4 : 474642 *bps*
- videoB1000k.mp4 : 925335 *bps*

```
<Representation id="1" mimeType="video/mp4" codecs="avc3.64000c" width="200" height="150" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="173339">
 <BaseURL>videoB_200_150_200k_dash.mp4</BaseURL>
```

Figura 14: *bps* necessário para o `videoB_200k.mp4`

```
<Representation id="2" mimeType="video/mp4" codecs="avc3.64001e" width="480" height="360" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="474642">
<BaseURL>videoB_480_360_500k_dash.mp4</BaseURL>
```

Figura 15: *bps* necessário para o `videoB_500k.mp4`

```
<Representation id="3" mimeType="video/mp4" codecs="avc3.64001e" width="640" height="480" frameRate="30" sar="1:1" startWithSAP="0" bandwidth="925355">
 <BaseURL>videoB_640_480_1000k_dash.mp4</BaseURL>
```

Figura 16: *bps* necessário para o `videoB_1000k.mp4`

Quanto à pilha protocolar envolvida no streaming adaptativo sobre HTTP, a Figura 17 demonstra todo o tráfego capturado, podendo assim tirar as seguintes conclusões:

- OSPF/IPv6 e OSPF/IPv4: Protocolos de roteamento em redes IPv6 e IPv4, que permitem que routers troquem informações de roteamento e determinem os melhores caminhos para o tráfego de dados.
- ARP: Protocolo de Resolução de Endereço, usado em redes Ethernet para mapear endereços IP em endereços MAC para a transmissão de pacotes de dados em redes locais.
- TCP/IP: Uma suíte de protocolos de rede que forma a base da internet, possibilitando a comunicação de dados e conectividade de rede de forma confiavél.

- HTTP: Protocolo de Transferência de Hipertexto, assente sobre TCP/IP e utilizado para solicitar e apresentar conteúdo da web.
- MP4/ISO-BMFF: Pacotes relacionados ao Formato de Arquivo de Mídia Base ISO (ISO-BMFF) dentro de arquivos MP4, contendo dados estruturados sobre o conteúdo multimédia.

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	5404	100.0	4769102	924 k	0	0	0
Ethernet	100.0	5404	1.6	75656	14 k	0	0	0
Internet Protocol Version 6	0.1	4	0.0	160	31	0	0	0
Open Shortest Path First	0.1	4	0.0	144	27	4	144	27
Internet Protocol Version 4	99.9	5398	2.3	107960	20 k	0	0	0
Transmission Control Protocol	99.5	5377	96.1	4584202	888 k	5366	4578908	887 k
Hypertext Transfer Protocol	0.2	9	28.3	1347318	261 k	7	2345	454
MP4 / ISOBMFF file format	0.0	2	28.2	1344563	260 k	2	1344973	260 k
Data	0.0	2	0.0	1368	265	2	1368	265
Open Shortest Path First	0.4	21	0.0	924	179	21	924	179
Address Resolution Protocol	0.0	2	0.0	56	10	2	56	10

Figura 17: Wireshark -> Protocol Hierarchy

Questão 3 - Ajuste o débito dos links da topologia de modo que o cliente no portátil Bela exiba o vídeo de menor resolução e o cliente no portátil Alladin exiba o vídeo com mais resolução. Mostre evidências.

De forma a verificar o débito adaptativo, o grupo estabeleceu um limite de largura de banda na conexão entre *sw2* e *Bela* com um valor um pouco a baixo do descrito no ficheiro `video_manifest.mpd` para o *videoB500k.mp4*.

A Figura 18 demontra esta conexão e encontra-se a vermelho pois está a ultrapassar o *threshold* imposto.

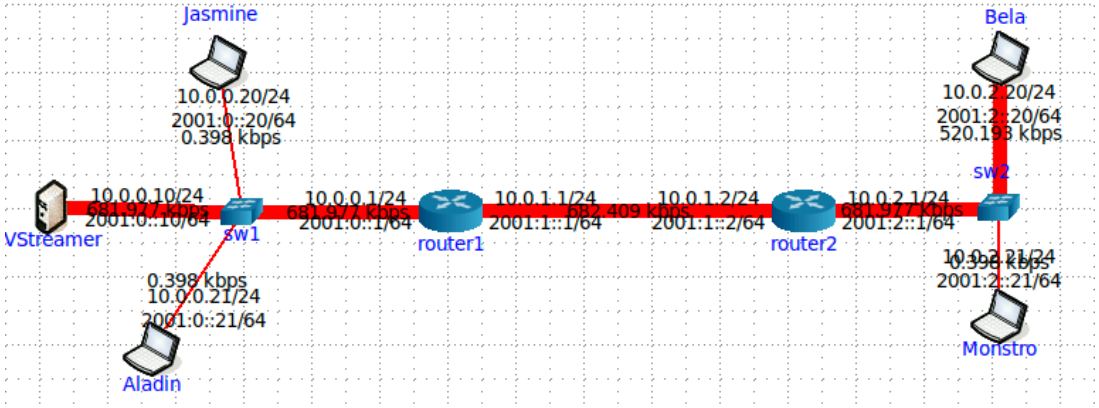


Figura 18: Tráfego do streaming *VStreamer* -> Bela

Inspecionando o tráfego do browser no cliente *Bela*, presente na Figura 19, verifica-se que após o pedido bem sucedido do `video_dash.html` e do `video_manifest.mpd` o cliente procede ao pedido dos ficheiros `.mp4`. Como não é bem sucedido no pedido do *videoB1000k.mp4*, pede um de largura de banda mais baixa, o *videoB500k.mp4*. Como este pedido também não é bem sucedido, pede o de largura de banda mais baixa, o *videoB200k.mp4*, sendo que este já é bem sucedido. Estes pedidos são orientados pelo ficheiro `video_manifest.mpd` que descreve o streaming adaptativo.

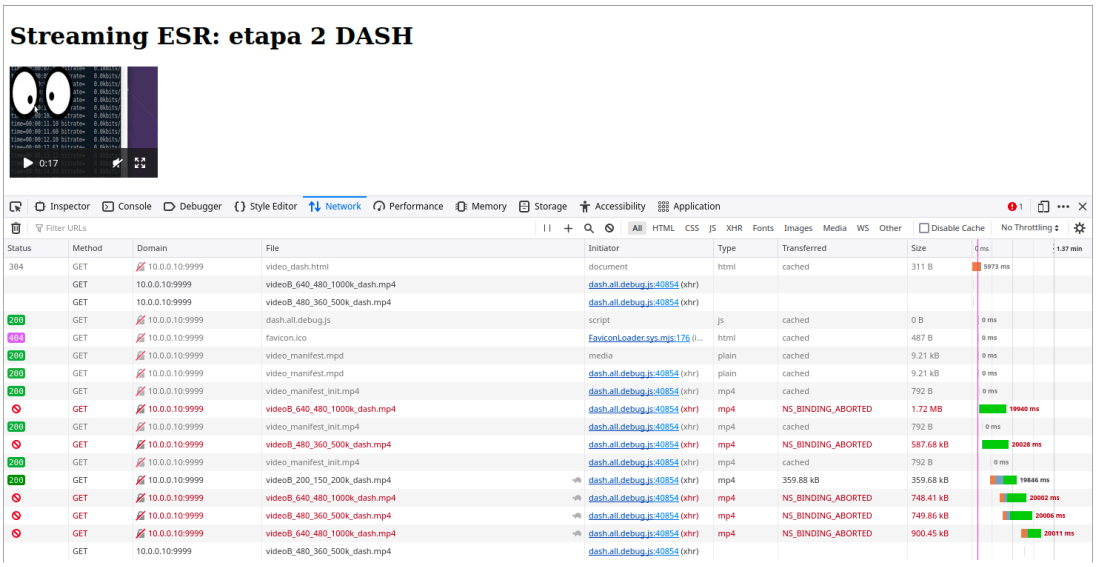


Figura 19: Análise de tráfego no Browser Bela

Para conseguir o video de maior resolução no browser do cliente *Alladin*, o grupo não estabeleceu qualquer limite na largura de banda. A Figura 20 demonstra o tráfego do browser e é possivel verificar que o primeiro pedido do vídeo é bem sucedido, logo não há necessidade de novos pedidos de menor resolução.

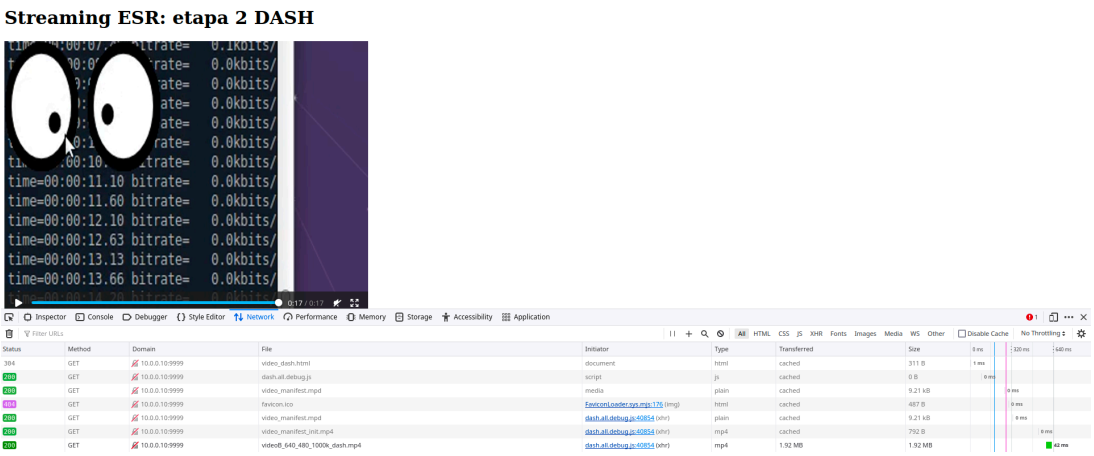


Figura 20: Análise de tráfego no Browser Alladin

A vantagem deste mecanismo é que em situações que a rede não suporta o streaming de uma certa resolução o browser adapta-se até conseguir streamar com a menor resolução possível - um caso real onde se observa este efeito é o Youtube, que suporta várias resoluções de streaming para o mesmo vídeo, de forma a oferecer uma solução flexível a todos os utilizadores.

<u>Questão 4</u> - Descreva o funcionamento do DASH neste caso concreto, referindo o papel do ficheiro MPD criado.

O DASH (*Dynamic Adaptive Streaming over HTTP*) e o ficheiro `.mpd` definem o sistema de streaming de vídeo DASH. O ficheiro fornece informações sobre a estrutura do vídeo, as suas representações e a adaptação com base nas condições de rede e dispositivos do espectador.

Neste caso, o efeito a ser estudado é a parte do ficheiro que define a Lógica de Adaptação. Esta parte esclarece como o servidor serve o conteúdo de acordo com as condições de rede e do cliente, como ilutra a Figura 21.

As Figuras 19 e 20 demonstram o efeito do sistema Dash em ação. As condições de rede na *Bela* não são suficientemente boas para a transmissão do video com maior resolução, desse modo, utilizando o ficheiro `.mpd` o cliente faz pedidos de vídeos com menor largura de banda, até que é bem sucedido no pedido do *videoB200k.mp4*

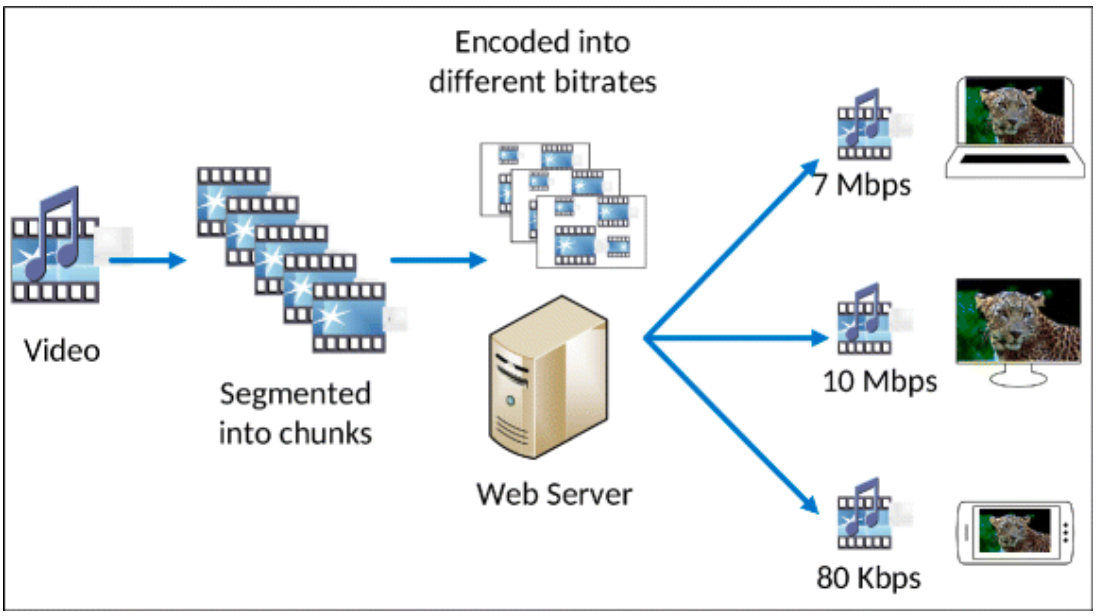


Figura 21: Diagrama do Funcionamento do Dash

2.3 | Etapa 3 - Streaming RTP/RTCP unicast sobre UDP e multicast com anúnicos SAP

Nesta etapa o grupo estudou o cenário unicast e o cenário multicast, que se tratam de duas abordagens diferentes para o encaminhamento de dados numa rede de computadores. Para comparar esses dois cenários, o grupo construiu duas streams de dados em redes diferentes, de forma a entender as vantagens e desvantagens em relação à escalabilidade e ao tráfego na rede.

Cenário Unicast

Para simular o cenário de unicasting foi utilizada a topologia de rede apresentada nas etapas anteriores, sendo inicialmente feito um teste de conetividade entre o VStreamer e a Bela e o VStreamer e a Jasmin, como mostra a Figura 22.

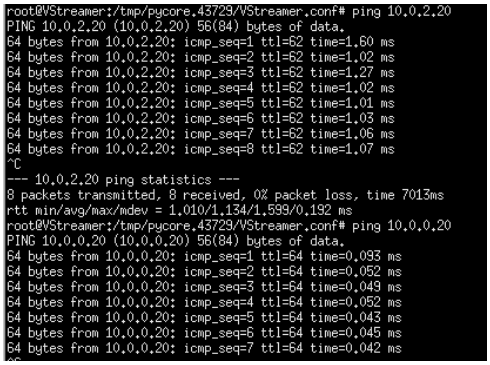
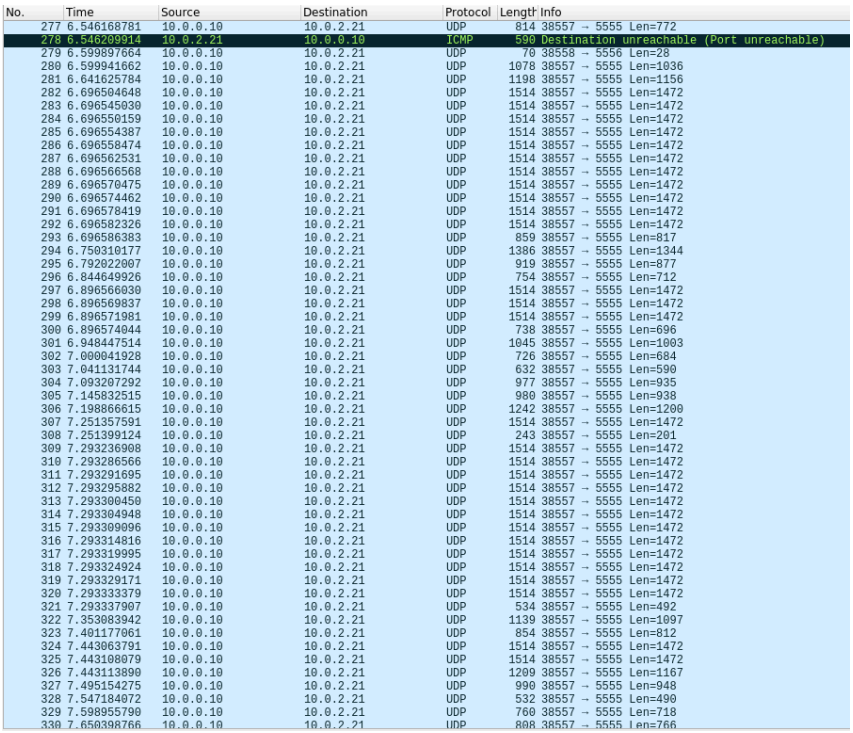


Figura 22: Teste de conetividade VStreamer -> Bela & Jasmin

Após verificado o correto funcionamento da rede, o grupo começa o streaming Unicast do VStreamer ao Monstro, sendo que foi capturado o seguinte conjunto de pacotes no VStreamer.

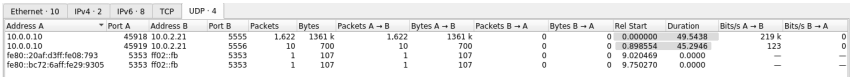


No.	Time	Source	Destination	Protocol	Length	Info
277	6.546168781	10.0.0.10	10.0.2.21	UDP	814	38557 → 5555 Len=772
278	6.546209914	10.0.2.21	10.0.0.10	ICMP	590	Destination unreachable (Port unreachable)
279	6.599897664	10.0.0.10	10.0.2.21	UDP	70	38558 → 5556 Len=28
280	6.599941662	10.0.0.10	10.0.2.21	UDP	1078	38557 → 5555 Len=1036
281	6.641625784	10.0.0.10	10.0.2.21	UDP	1198	38557 → 5555 Len=1156
282	6.696504648	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
283	6.696545030	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
284	6.696550159	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
285	6.696554387	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
286	6.696558474	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
287	6.696562531	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
288	6.696566568	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
289	6.696570475	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
290	6.696574462	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
291	6.696578419	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
292	6.696582326	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
293	6.696586383	10.0.0.10	10.0.2.21	UDP	859	38557 → 5555 Len=817
294	6.750310177	10.0.0.10	10.0.2.21	UDP	1386	38557 → 5555 Len=1344
295	6.792022007	10.0.0.10	10.0.2.21	UDP	919	38557 → 5555 Len=877
296	6.844649926	10.0.0.10	10.0.2.21	UDP	754	38557 → 5555 Len=712
297	6.896566030	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
298	6.896569837	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
299	6.896571981	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
300	6.896574044	10.0.0.10	10.0.2.21	UDP	738	38557 → 5555 Len=696
301	6.948447514	10.0.0.10	10.0.2.21	UDP	1045	38557 → 5555 Len=1003
302	7.000041928	10.0.0.10	10.0.2.21	UDP	726	38557 → 5555 Len=684
303	7.041131744	10.0.0.10	10.0.2.21	UDP	632	38557 → 5555 Len=590
304	7.093207292	10.0.0.10	10.0.2.21	UDP	977	38557 → 5555 Len=935
305	7.145832515	10.0.0.10	10.0.2.21	UDP	980	38557 → 5555 Len=938
306	7.198866615	10.0.0.10	10.0.2.21	UDP	1242	38557 → 5555 Len=1200
307	7.251357591	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
308	7.251399124	10.0.0.10	10.0.2.21	UDP	243	38557 → 5555 Len=201
309	7.293236908	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
310	7.293286566	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
311	7.293291695	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
312	7.293295882	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
313	7.293300450	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
314	7.293304948	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
315	7.293309096	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
316	7.293314816	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
317	7.293319995	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
318	7.293324924	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
319	7.293329171	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
320	7.293333379	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
321	7.293337907	10.0.0.10	10.0.2.21	UDP	534	38557 → 5555 Len=492
322	7.353083942	10.0.0.10	10.0.2.21	UDP	1139	38557 → 5555 Len=1097
323	7.401177061	10.0.0.10	10.0.2.21	UDP	854	38557 → 5555 Len=812
324	7.443063791	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
325	7.443108079	10.0.0.10	10.0.2.21	UDP	1514	38557 → 5555 Len=1472
326	7.443113890	10.0.0.10	10.0.2.21	UDP	1209	38557 → 5555 Len=1167
327	7.495154275	10.0.0.10	10.0.2.21	UDP	990	38557 → 5555 Len=948
328	7.547184072	10.0.0.10	10.0.2.21	UDP	532	38557 → 5555 Len=490
329	7.598955790	10.0.0.10	10.0.2.21	UDP	760	38557 → 5555 Len=718
330	7.650398766	10.0.0.10	10.0.2.21	UDP	808	38557 → 5555 Len=766

Figura 23: Captura Wireshark do VStreamer em Unicast

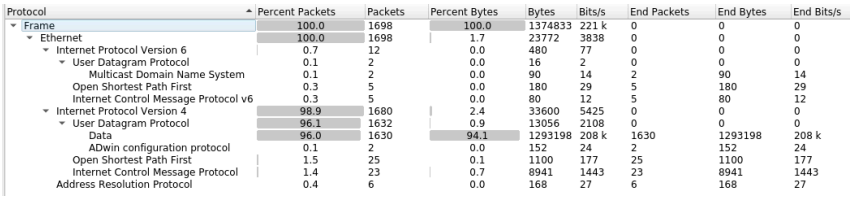
Como é possivel ver na Figura 24, no cenário Unicast cria-se um fluxo de dados com o cliente-alvo.

Na Figura 25 é possível ver os protocolos envolvidos na transmissão, sendo o protocolo de transporte o UDP, ao contrário das etapas anteriores.



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.0.10	45918	10.0.2.21	5555	1.622	1361 k	1.622	1361 k	0	0	0.000000	49.5438	219 k	0
10.0.0.10	45919	10.0.2.21	5556	10	700	10	700	0	0	0.898554	45.2946	123	0
fe80::20af:d3ff:fe08:793	5353	ff02::fb	5353	1	107	1	107	0	0	9.020469	0.0000	—	—
fe80::bc72:6aff:fe29:9305	5353	ff02::fb	5353	1	107	1	107	0	0	9.750270	0.0000	—	—

Figura 24: Fluxos Gerados em Unicast



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	1698	100.0	1374833	221 k	0	0	0
Ethernet	100.0	1698	1.7	23772	3838	0	0	0
Internet Protocol Version 6	0.7	12	0.0	480	77	0	0	0
User Datagram Protocol	0.1	2	0.0	16	2	0	0	0
Multicast Domain Name System	0.1	2	0.0	90	14	2	90	14
Open Shortest Path First	0.3	5	0.0	180	29	5	180	29
Internet Control Message Protocol v6	0.3	5	0.0	80	12	5	80	12
Internet Protocol Version 4	98.9	1680	2.4	33600	5425	0	0	0
User Datagram Protocol	96.1	1632	0.9	13056	2108	0	0	0
Data	96.0	1630	94.1	1293198	208 k	1630	1293198	208 k
ADwin configuration protocol	0.1	2	0.0	152	24	2	152	24
Open Shortest Path First	1.5	25	0.1	1100	177	25	1100	177
Internet Control Message Protocol	1.4	23	0.7	8941	1443	23	8941	1443
Address Resolution Protocol	0.4	6	0.0	168	27	6	168	27

Figura 25: Protocolos envolvidos no Unicast

Cenário Multicast

Para simular o cenário de multicasting foi criada uma nova topologia de rede. De notar que a nova topologia coloca todos os dispositivos, clientes e servidor, dentro da mesma rede local.

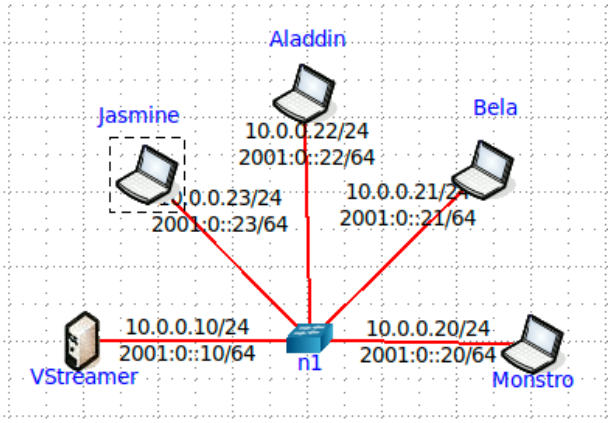


Figura 26: Topologia de Rede Multicast

Para testar a nova topologia de rede, o grupo fez um teste de conetividade e verificou que estava tudo operacional.

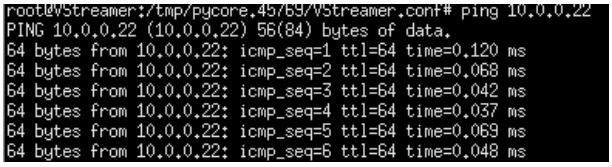
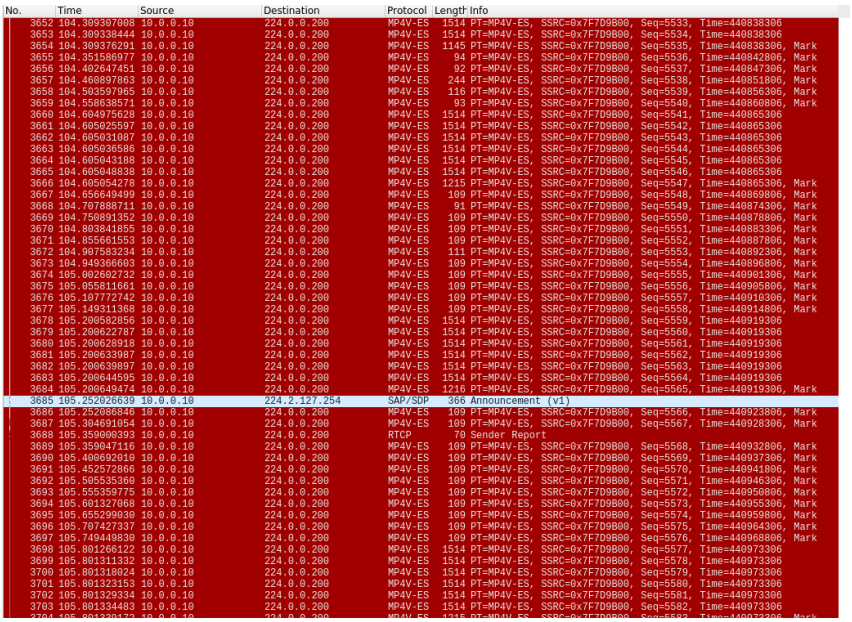


Figura 27: Teste de Conetividade da Rede Multicast

De seguida iniciou-se o streaming de dados em Multicast, sendo a Figura 28 o tráfego capturado na saída do VStreamer.

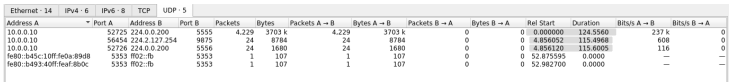


No.	Time	Source	Destination	Protocol	Length	Info
3652	104.309307008	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5533, Time=440838306
3653	104.309338444	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5534, Time=440838306
3654	104.309376291	10.0.0.10	224.0.0.200	MP4V-ES	1145	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5535, Time=440838306, Mark
3655	104.351586977	10.0.0.10	224.0.0.200	MP4V-ES	94	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5536, Time=440842806, Mark
3656	104.402647451	10.0.0.10	224.0.0.200	MP4V-ES	92	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5537, Time=440847306, Mark
3657	104.460897863	10.0.0.10	224.0.0.200	MP4V-ES	244	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5538, Time=440851806, Mark
3658	104.503597965	10.0.0.10	224.0.0.200	MP4V-ES	116	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5539, Time=440856306, Mark
3659	104.558638571	10.0.0.10	224.0.0.200	MP4V-ES	93	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5540, Time=440860806, Mark
3660	104.604975628	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5541, Time=440865306
3661	104.605025597	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5542, Time=440865306
3662	104.605031087	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5543, Time=440865306
3663	104.605036586	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5544, Time=440865306
3664	104.605043188	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5545, Time=440865306
3665	104.605048838	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5546, Time=440865306
3666	104.605054278	10.0.0.10	224.0.0.200	MP4V-ES	1215	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5547, Time=440865306, Mark
3667	104.656649499	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5548, Time=440869806, Mark
3668	104.707888711	10.0.0.10	224.0.0.200	MP4V-ES	91	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5549, Time=440874306, Mark
3669	104.750891352	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5550, Time=440878806, Mark
3670	104.803841855	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5551, Time=440883306, Mark
3671	104.855661553	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5552, Time=440887806, Mark
3672	104.907583234	10.0.0.10	224.0.0.200	MP4V-ES	111	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5553, Time=440892306, Mark
3673	104.949366603	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5554, Time=440896806, Mark
3674	105.002602732	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5555, Time=440901306, Mark
3675	105.055811661	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5556, Time=440905806, Mark
3676	105.107772742	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5557, Time=440910306, Mark
3677	105.149311368	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5558, Time=440914806, Mark
3678	105.200582856	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5559, Time=440919306
3679	105.200622787	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5560, Time=440919306
3680	105.200628918	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5561, Time=440919306
3681	105.200633987	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5562, Time=440919306
3682	105.200639897	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5563, Time=440919306
3683	105.200644595	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5564, Time=440919306
3684	105.200649474	10.0.0.10	224.0.0.200	MP4V-ES	1216	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5565, Time=440919306, Mark
3685	105.252026639	10.0.0.10	224.2.127.254	SAP/SDP	366	Announcement (v1)
3686	105.252086846	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5566, Time=440923806, Mark
3687	105.304691054	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5567, Time=440928306, Mark
3688	105.359000393	10.0.0.10	224.0.0.200	RTCP	70	Sender Report
3689	105.359047116	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5568, Time=440932806, Mark
3690	105.400692010	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5569, Time=440937306, Mark
3691	105.452572866	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5570, Time=440941806, Mark
3692	105.505535360	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5571, Time=440946306, Mark
3693	105.555359775	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5572, Time=440950806, Mark
3694	105.601327068	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5573, Time=440955306, Mark
3695	105.655299030	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5574, Time=440959806, Mark
3696	105.707427337	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5575, Time=440964306, Mark
3697	105.749449830	10.0.0.10	224.0.0.200	MP4V-ES	109	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5576, Time=440968806, Mark
3698	105.801266122	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5577, Time=440973306
3699	105.801311332	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5578, Time=440973306
3700	105.801318024	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5579, Time=440973306
3701	105.801323153	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5580, Time=440973306
3702	105.801329334	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5581, Time=440973306
3703	105.801334483	10.0.0.10	224.0.0.200	MP4V-ES	1514	PT=MP4V-ES, SSRC=0x7F7D9B00, Seq=5582, Time=440973306

Figura 28: Captura Wireshark do VStreamer em Unicast

Como é possivel ver na Figura 29, no cenário Multicast não se cria um fluxo de dados com o cliente-alvo, mas sim uma transmissão para um endereço de grupo, do qual vários clientes podem "escutar".

Na Figura 30 é possível ver os protocolos envolvidos na transmissão Multicast, sendo utilizado o protocolo de transporte UDP com SAP (Session Announcement Protocol) e RTP (Real-time Transport Protocol) / RTCP (RTP Control Protocol).



Ethernet · 14 | IPv4 · 6 | IPv6 · 8 | TCP | UDP · 5

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.0.10	52725	224.0.0.200	5555	4.229	3703 k	4.229	3703 k	0	0	0.000000	124.5560	237 k	0
10.0.0.10	56454	224.2.127.254	9875	24	8784	24	8784	0	0	4.856052	115.4968	608	0
10.0.0.10	52726	224.0.0.200	5556	24	1680	24	1680	0	0	4.856120	115.6005	116	0
fe80::b45c:10ff:fe0a:89d8	5353	ff02::fb	5353	1	107	1	107	0	0	52.875595	0.0000	—	—
fe80::b493:40ff:feaf:8b0c	5353	ff02::fb	5353	1	107	1	107	0	0	52.982700	0.0000	—	—

Figura 29: Fluxos Gerados em Multicast

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	4303	100.0	3715944	238 k	0	0	0
Ethernet	100.0	4303	1.6	60242	3869	0	0	0
Internet Protocol Version 6	0.2	8	0.0	320	20	0	0	0
User Datagram Protocol	0.0	2	0.0	16	1	0	0	0
Multicast Domain Name System	0.0	2	0.0	90	5	2	90	5
Internet Control Message Protocol v6	0.1	6	0.0	96	6	6	96	6
Internet Protocol Version 4	99.8	4295	2.3	85972	5521	0	0	0
User Datagram Protocol	99.4	4277	0.9	34216	2197	0	0	0
Session Announcement Protocol	0.6	24	0.2	7776	499	0	0	0
Session Description Protocol	0.6	24	0.2	7200	462	24	7200	462
Real-Time Transport Protocol	93.8	4038	89.9	3341012	214 k	0	0	0
MP4V-ES	93.8	4038	88.6	3292556	211 k	4038	3292556	211 k
Real-time Transport Control Protocol	0.6	24	0.0	672	43	24	672	43
Data	4.4	190	5.0	185176	11 k	190	185176	11 k
ADwin configuration protocol	0.0	1	0.0	52	3	1	52	3
Internet Group Management Protocol	0.4	18	0.0	304	19	18	304	19

Figura 30: Protocolos envolvidos no Multicast

O grupo testou o cenário multicast até quatro clientes em simultaneo, como se vê na Figura 31. E como se verifica na Figura 29, a quantidade de fluxos gerados não aumentou proporcionalmente

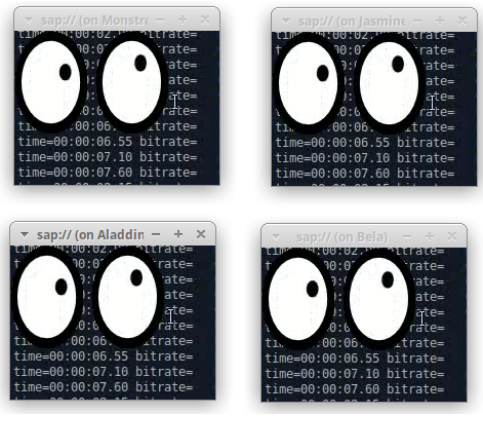


Figura 31: Clientes em transmissão Multicast

Questão 5 - Compare o cenário unicast aplicado com o cenário multicast. Mostre vantagens e desvantagens na solução multicast ao nível da rede, no que diz respeito a escalabilidade (aumento do nº de clientes) e tráfego na rede. Tire as suas conclusões.

Cenário Unicast:

No cenário unicast, cada pacote de dados é enviado individualmente de uma fonte para um destino específico. Isso significa que, se houver múltiplos clientes para os mesmos dados, a fonte terá que enviar uma cópia separada para cada cliente.

Vantagens:

- Precisão: Cada destinatário recebe exatamente o que precisa, e não mais. Isso é útil quando a precisão é crítica, como em videoconferências ou transferências de arquivos privados.
- Controle: É fácil gerenciar e rastrear o tráfego de dados para cada destinatário, tornando-o adequado para redes pequenas ou com requisitos específicos de segurança.

Desvantagens:

- Escalabilidade: À medida que o número de clientes aumenta, o tráfego na rede cresce exponencialmente, já que cada destinatário recebe sua própria cópia dos dados. Isso pode sobrecarregar a rede e a fonte.
- Ineficiência: Consome largura de banda de forma ineficiente, especialmente quando há muitos destinatários para os mesmos dados, resultando em congestionamentos e latências.

Cenário Multicast:

No cenário multicast, um único pacote de dados é enviado para um grupo de destinatários que compartilham um interesse comum nos dados. Isso significa que a fonte envia apenas uma cópia dos dados, independentemente do número de clientes.

Vantagens:

- Eficiência: Reduz o tráfego na rede, já que apenas uma cópia dos dados é enviada, independentemente do número de destinatários. Isso economiza largura de banda e reduz o congestionamento da rede.
- Escalabilidade: É mais escalável do que o unicast, pois lida de forma eficiente com grupos grandes de clientes sem sobrecarregar a fonte ou a rede.
- Tempo Real: Útil para transmissões em tempo real, como streaming de vídeo, onde vários espectadores assistem ao mesmo conteúdo ao mesmo tempo.

Desvantagens:

- Complexidade: A implementação de multicast pode ser mais complexa do que o unicast, exigindo roteadores e switches que suportem multicast e configurações específicas de grupos.
- Precisão Limitada: Não é tão preciso quanto o unicast, já que todos os membros do grupo recebem os mesmos dados, independentemente de sua necessidade específica.

Concluíndo, em termos de escalabilidade, o multicast tem uma vantagem clara, pois é mais eficiente ao lidar com um grande número de destinatários em comparação com o unicast. Para redes com necessidades de precisão e controle mais rígidos, o unicast pode ser preferível.

3 | Conclusões

Na Etapa 1, onde exploramos o Streaming HTTP Simples, observamos uma escalabilidade linear na largura de banda conforme o número de clientes aumenta. Contudo, ficou evidente que o protocolo utilizado não é altamente escalável devido à sua natureza orientada à conexão, o que implica a criação de novas portas para cada cliente, resultando em limitações.

Na Etapa 2, focamos no Streaming Adaptativo usando MPEG-DASH. Aqui, compreendemos que o streaming adaptativo desempenha um papel crucial, permitindo ajustar dinamicamente a qualidade do vídeo com base nas condições da rede e dos dispositivos dos usuários finais. O arquivo MPD (Media Presentation Description) revelou-se essencial, capacitando os clientes a fazer escolhas informadas sobre a qualidade do vídeo, proporcionando uma experiência de usuário otimizada.

Na Etapa 3, ao investigar o Streaming Unicast e Multicast, destacamos que o Multicast se destaca como uma solução altamente eficiente em termos de tráfego de rede e escalabilidade. Ao enviar apenas uma cópia dos dados para um grupo de destinatários, o Multicast oferece uma abordagem economicamente eficaz e escalável. No entanto, o Unicast permanece indispensável nos cenários onde a precisão e o controle são fundamentais, embora implique um consumo de largura de banda mais substancial.

Em síntese, cada abordagem de streaming oferece vantagens específicas, sendo a escolha entre elas influenciada pelos requisitos exclusivos de cada cenário. Este tipo de conclusões são fundamentais para se poder tomar decisões mais informadas em projetos de streaming futuros.